Division 6 — Lincoln Laboratory
Massachusetts Institute of Technology
Lexington 73, Massachusetts

SUBJECT: TX-0 DIRECT INPUT UTILITY SYSTEM

To: Distribution List

From: John T. Gilmore, Jr.

Date: 3 October 1958

Approved: JTG

ABSTRACT: Large high speed memories have arrived. Many of our present procedures were born of necessity because of limited storage. It is time to review our present techniques and philosophies in planning and programming computer applications. At present we write programs which perform complicated and monotonous tasks efficiently and rapidly. But writing and debugging these programs are also complicated and monotonous tasks. Now that we have larger memories, why not try writing programs which will help us with our dirty work? This memo describes a utility system which is basic in form but unique in the sense that it assists the programmer in debugging and modifying his program at the console. This is accomplished by moving the utility system into memory alongside the program to be debugged and providing direct communication between the utility system and the programmer.

Distribution List:

Group 63 Staff
Bagley, P. R.
Heart, F.
Mayer, R.
Dinneen, G. P.
Ziegler, H.
Vanderburgh, A.
Arden, Dean (Barta)
Attridge, W.
Hazel, F. P.
Hosier, W.
Grandy, C.
Buzzard, R.
Holmes, L.
Bennington, H. (Rand)
Margaret Clark (Rand)
Burrows, J. H.
Collins, L. B.
Goldberg, S.
Jones, N.T.
Marnie, G. L.
Daggett, N.
Arnow, J.
Zraket, C. A.
Israel, D.
Rich, E.
Bailey, D.
Rising, H. K.
Frachtman, H. E.
Vance, R. R.
Neeb, Donna (Rand)
Thomas, L. M.
Dustin, D. E.
Tritter, A. L.
Briscoe, H.
Carma Darley
Joseph Thompson (Rand)
Paddock, R. B.
Pughe, E. W.
Rowe, A.
Zraket, C. A.

TABLE OF CONTENTS

INTRODUCTION

TX-0 is an experimental computer which was built to test transistor circuitry and a 256x256 magnetic core memory. Its terminal equipment consists of a photo-electric paper tape reader, a paper tape punch, an oscilloscope, and an input-output Flexowriter typewriter. Additional equipment such as magnetic tape, high speed printers, etc., will be added at a later date. Since it is a test computer, most of the computer time is devoted to marginal checking and the operation of test programs. However, the fact that such a large internal memory of 65,536 registers now exists prompted us to try some new programming techniques in the time available between test operations.

One technique was to use the bulk of the memory as a secondary storage medium instead of magnetic drums or magnetic tapes. Wes Clark wrote a conversion program using this idea. The result was a fast conversion process and one which could translate a flexible symbolic language. For example, address tags, address sections of instructions, and constants can now be expressed as English words as well as code letters and numerals. The conversion program has no tape scanning problems and can look at the whole program to be converted as many times as necessary in a very short period of time. A memo describing the rules of the conversion program's vacabulary is being written by Wes and should be available soon.

Another valuable technique which will be described in this memo is that of moving a complete utility system into memory (see Figure 1) as well as the program (or programs) to be operated and debugged. Using the input-output Flexowriter typewriter as a means of communication with the utility system, debugging at the console should be reasonably simple and convenient.

Debugging at the console is not new. In fact, it is the oldest form of debugging. For the past few years it has been regarded as a great professional sin, but like most sins, it still exists, is rarely talked about, and hardly ever admitted. High costs and inefficiency

are the main reasons why it is condemned. At the present time most computers rent for approximately three hundred dollars per hour. The time saved by debugging at the console of the computer is usually far less valuable than the cost of the additional computer time. The second reason is that at present, it is extremely difficult to get an objective view of what a program is doing or has done while at the console. An attempt to find out usually requires manual switch settings, limited examinations by push buttons, or trace program data which is not immediately available. A parallel argument is that thinking under pressure promotes poor reasoning and lots of bad guesses. The general result is wasted time----debugging time as well as precious computer time.

In the face of these arguments, why try to encourage or develop techniques for debugging at the console? Because in certain situations and applications the time between the instant that the first instruction is written and the moment that the results are produced, are of the utmost importance. In such cases any time is more than worth the additional rental cost. Data Processing, Operational Research, and Real Time problems are rapidly producing more and more situations like this. Although these critical situations will involve only a small percentage of the programs being operated on a given computer, utility systems must be avilable to handle them.

There is still the problem of the programmer thinking on his feet, trying to find what went wrong with his program and how to modify it. It is this phase in which we are extremely interested.

At the present time most utility programs do not provide enough immediate information to enable a programmer to find his erroe while he is still at the console. And if he can find it, few systems are capable of allowing him to make an immediate change without the proparation of a card, paper tape, or some other secondary input medium,

especially if he wants to retain the symbolic language of the conversion program. Another handicap is that he cannot examine any given part of his program without having previously prepared a request card or tape. And finally, the use of trace programs is desirable only to the extent that they are simple enough to invite their use by the programmer. A tracing program that can give a visual picture of what a program is doing is of much more value than one which produces reams of printed information.

II DESCRIPTIONS OF INDIVIDUAL ROUTINES

Our first and most important reason for writing a direct utility system was to determine what routines would be most useful in assisting a programmer at the console. We therefore began with a minimum of specifications. The programming language was restricted to absolute addressed instructions and constants. This does not prevent a TX-0 programmer from writing his original program in relative form. The conversion program written by Wes Clark has a very flexible language and permits one to obtain an absolute addressed translation of a program which has been written in relative form. Later, when we have determined what the ideal direct utility system should contain we will include a more flexible programming language.

At the present time the system contains over a dozen routines. Some of them are very necessary and others provide services whose only importance lies in the fact that they reduce impatience on the part of the programmer. Each routine obtains its required information by asking specific English questions on the console typewriter. The answers are typed on the same typewriter by the programmer and may be English words or numbers depending on the nature of the question asked.

Expedite

The system was designed around a master dispatch routine called EXPEDITE. Its job is to read the Flexowriter coded symbols coming from the typewriter and recognize English words. There is a small English vocabulary in the system and each stored word has an instruction associated with it. This instruction tells EXPEDITE what to do once an incoming

word has been recognized. Most of the words are the names of the other routines in the system and the associated instructions inform EXPEDITE where the routines start. EXPEDITE not only acts as a dispatcher but is also used by the other routines as a subroutine when they require English answers from the console. Whenever a word is received which is not listed in the vocabulary, EXPEDITE informs the typist of the error. Also, should EXPEDITE receive its own name it will print out the words in its vocabulary.

Hark

The conversion program of the system is called HARK. Using HARK the programmer is able to call for specific registers for examination and, if necessary, modification. When an examination is requested, HARK translates the binary contents of the requested register into an octal addressed instruction. If the register contains the binary form of an operate-class command for which there is a three letter mnemonic code, HARK will type out the three letters which represent that operate-class command. When a modification is desired, the programmer may type the change as an octal instruction, a coded operate-class command, an octal number, or any combination of these three as long as they are separated by plus or minus signs. (Part IV provides explicit typing instructions.) Certain control keys of the typewriter allow a programmer to examine and modify a series of registers very rapidly. When it is desirable, HARK will provide a binary punched record of the modifications (in Read-In Mode format). HARK is also capable of recognizing a request to transfer control to any register in the program being debugged. When the programmer is through with HARK, the Stop Code button is pressed and EXPEDITE is ready to receive the name of another requested routine.

Prince

Every utility system must have a storage print-out routine. Ours is called PRINCE. It requires three pieces of information from the programmer. They are:

1. The kind of page layout.
2. How the binary contents of storage are to be translated.
3. What range of storage is to be translated and printed.

To determine the kind of page layout PRINCE types, DO YOU WANT VERTICAL COLUMN LAYOUT. The answer "YES" informs PRINCE that each printed page is to contain two columns of words. Each column will be 100 (octal) words long and the address sequence will run down the left column and then down the right. PRINCE will also lay out each page so that the address of the first word position in the left column will have a factor of 100 (octal). If the first word to be printed does not have an address with a factor of 100, PRINCE will print the first word in the left column relative to its position from the last address with a factor of 100. For example, if the address of the first word to be printed is 507, PRINCE will skip the first seven positions in the left column before printing the contents of register 507. In this way each page is laid out in a uniform manner. If "NO" is the answer to the first question, PRINCE will lay out 10 (octal) words to a line and 2^9 words per page. Although the vertical column layout printing time is longer than the horizontal, it is more popular because it has room for comments and is laid out in the way in which most programs are written. Since PRINCE has only two kinds of page layout the answers YES or NO are sufficient. If more layouts had been available the question would have been reworded and the answer would have required specific English word(s); e.g., DOUBLE VERTICAL, SINGLE VERTICAL, HORIZONTAL, etc.

DO YOU WANT OCTAL INSTRUCTIONS is the next question. "YES", is obvious. "NO", will inform PRINCE that each register in the range is to be translated as a constant.

The range is determined by the questions, WHAT IS THE FIRST ADDRESS TO BE PRINTED, and WHAT IS THE LAST ADDRESS. An octal address is required by the programmer after each question has been typed. It is interesting to note that PRINCE uses EXPEDITE as a subroutine when it requires an English answer and HARK when it requires a numerical answer.

Following the reception of the last address to be printed, the range is printed out in the desired form. When the range has been printed PRINCE asks, IS THERE MORE TO BE PRINTED. "YES", returns control to the

first question again. "NO", returns to EXPEDITE.

Punchy

In order to allow the programmer to retain a corrected binary record of his program, we have included a storage punch-out routine called PUNCHY. The first question asked by PUNCHY is, DO YOU WANT A TITLE. "YES". will cause the question, WHAT IS THE TITLE. The programmer then types what he wants labeled on the beginning of his tape. For example, SAMPLE TEST PROGRAM 25 FEB. 57. "NO", will bypass the title question and lead to the question that follows the title punch-out, namely, DO YOU WANT THE NORMAL INPUT ROUTINE LAYOUT. "YES", will remind PUNCHY to punch out the Input Routine on tape in Read-In Mode format before punching the requested range of storage in the normal Input Routine format. "NO", will inform PUNCHY that the range is to be punched out in Read-In Mode format. A FUNCTIONAL DESCRIPTION OF THE TX-0 COMPUTER (6M-4789), explains the difference between the two kinds of binary layout. The Read-In Mode format is better for short ranges. The next two questions ask for the first and last address of the range to be punched out. The range is then punched out and then the question, ARE THERE MORE BLOCKS TO BE PUNCHED OUT, is asked. "YES", returns control to the questions asking for range addresses. "NO", causes the question, WHAT IS THE ADDRESS OF THE STARTING INSTRUCTION, to be asked. After an address has been typed by the programmer, the question, DO YOU WANT THE COMPUTER TO STOP AFTER READING IN THIS PUNCHED PROGRAM, is typed. "YES", informs PUNCHY to punch out the starting instruction in a form which will cause the Read-In Mode or the Input Routine to stop the computer before transferring control to the indicated address. "NO", will cause the opposite. Once the YES or NO has been received by PUNCHY, the starting instruction is punched out in the desired form and control is returned to EXPEDITE.

R And Me

The system contains a read-in routine similar to the Input Routine described in 6M-4789. When EXPEDITE receives an R followed by a carriage return, control is transferred to the input routine. Once R has read in a tape, the routine will print either GO TO of SUM ERROR, depending on

whether or not the tape was read in correctly. In either case control will be transferred back to EXPEDITE so that a programmer can call for any one of the various routines in the system before operating his program. When he is ready to operate his program, the word ME will tell EXPEDITE to transfer control to the starting instruction which was on the end of the last tape read in by R.

Surprise

The SURPRISE routine is a useful one which reviews all the registers of a given program and types out the addresses of those registers whose contents have changed from their original form. It is called SURPRISE because the information it produces usually comes as a big surprise to the programmer.

After a program has operated and erred, the binary tape is again placed in the photoelectric reader. EXPEDITE is given control and the word SURPRISE is typed by the programmer. The tape is then read in by SURPRISE and the address of each register whose contents have changed is typed out along with the present contents and the original contents (translated as octal instructions or coded operate class commands). The original contents are restored in each case and when the tape review has been completed, control is transferred back to EXPEDITE.

Find

FIND is a routine which was born when we started to combine several programs in storage. We found that redundancy and confusion resulted when two or more programs were formed in memory. Areas which appeared to be empty were being used by other programs for temporary storage and programs were killing each other by store instructions or just strolling through certain crucial routines. We wrote FIND to help us find and avoid these difficulties. Its usefulness to a programmer probably depends on the length of his program and the number of small routines used to build it. FIND begins by typing the question, DO YOU WANT TO FIND A WORD, REFERENCES TO AN ADDRESS, OR SEARCH FOR AN OUTLAW TRN INSTRUCTION. The three possible answers are WORD, ADDRESS, and OUTLAW.

Word - When WORD is typed, FIND returns with, WHAT IS THE WORD. The answer may be an octal instruction, a coded operate-class command, an octal number, or any combination of the three as long as they are separated by plus or minus signs. FIND then searches the whole of memory (20 seconds) and types out the addresses of those registers which contain that word. When all of memory has been examined, FIND returns control to EXPEDITE. If nothing is found, FIND will print NO ADDRESS before returning control to EXPEDITE.

Address - When ADDRESS is typed by the programmer, FIND returns with, WHAT IS THE ADDRESS. The answer must be an octal number. FIND searches the whole of memory and types out the addresses and contents of those registers whose address sections agree with the address specified by the programmer. When all of memory has been examined, FIND will return control to EXPEDITE.

Outlaw- Have you ever had a program stop in some section of memory that was not even part of your program---or worse still, found it sitting still in a block of registers which contained constants instead of instructions? How it got there is a good question, and usually a hard one to answer, since it probably did not come to a complete stop as soon as it left your program. When OUTLAW is typed by the programmer, FIND returns with, WHERE DID YOUR PROGRAM STOP. An octal number is required. As soon as FIND receives the address it examines that register and those that precede it. It doubles back through storage until it finds an instruction or set of instructions which would cause an unconditional control transfer. The address given it by the programmer and the address following the unconditional transfer constitute an area which cannot be penetrated by control except by the use of a transfer control instruction. FIND then searches the whole of memory and examines each transfer instruction's address section. If the address section's value falls within the area in question, the address of the register and the transfer instruction will be printed out. When all of memory has been examined, FIND returns control to EXPEDITE. If nothing is found, NO ADDRESS will be printed

before returning control to EXPEDITE.

Flow Chart Display

We mentioned earlier that we felt it was desirable to have a trace program which was able to give a programmer a visual picture of what his program was doing while he was at the console. The system has a routine which does this in a limited way. It is rather primitive but it may give an idea on how to accomplish the desired result. We call it The Flow Chart Display Subroutine. It performs two functions, the first of which is to display the four sides of a given box in a given area of the display scope. The second function is to keep a record of the sequence in which requests were made for all boxes. When the subroutine is requested the main program must supply three pieces of information, namely:

1. The x,y coordinates of the lower left corner of the box to be displayed.
2. The width and height of the box.
3. The Flexowriter code for some letter or numeral which the subroutine can use to identify this specific box.

The purpose of the subroutine is to indicate the control path in a program to be debugged. This is accomplished by first dividing the program into sections and drawing a flow diagram in which each section is represented as a box. The diagram must be drawn on transparent paper superimposed on a TX-0 Octal Graph Chart (see figures 3 and 4). When the diagram is completed the programmer notes the coordinates and dimensions of each section's box by referring to the Octal Graph Chart. In writing his program, he will insert a transfer control instruction to the Flow Chart Display Subroutine followed by the necessary information before each section's list of instructions. When the program is ready to be operated, the transparent copy of the flow diagram is placed on the face of the oscilloscope. During the operation of the program, as each section is performed, its corresponding box on the flow diagram will be illuminated. In this way the programmer will be able to follow the control path through his program. The number of times a box is illuminated for one operation of the section it represents, can be controlled externally

(by means of the toggle switch accumulator). This allows the programmer to control the speed of his picture or to shut it off completely.

Path

When the program has stopped, the programmer may obtain a printed record of the sequence in which the sections were performed. This is done by transferring control to EXPEDITE and typing the word PATH. The result is a printed sequence of alpha numeric symbols (81 per line). When the print-out is completed, FINISHED is typed by PATH, and control is returned to EXPEDITE.

As an example, the boxes on the flow chart of figure 4 were lettered a,b,c,d,e,f,g,h,l,r,m, and s. A printed sequence from PATH might appear as:

abcdrmgcdrmgcdrmgcldrmgcldrmgcdefgcdefs
finished

Be Brief

Once a programmer has become familiar with the questions of each routine there is no need to continue the lengthy wording of each question. When BE BRIEF is typed by the programmer EXPEDITE will reduce each question in every routine to one or two words. When it has completed this task and is ready to receive title requests again, it informs the programmer by typing YES SIR.

III SUBROUTINES

In writing the Direct Input Utility System we made an effort to incorporate as many subroutines as possible. This was done for two reasons. First, it makes the job of modifying the system much easier, and second, it privides a few subroutines that a programmer can use in his own program.

Each subroutine has been written so that it must be entered with a transfer control instruction in the accumulator. The address section

of this instruction should contain the address of the register where control is to be returned when the subroutine has completed its task. Unfortunately, TX-0 is such a simple machine that it does not have the equivalent of a Whirlwind A Register. Therefore, it is necessary to program the return address when using subroutines. But since the computer and this system only exist to discover new potentials from large memories, we have no reason to complain. TX-2, the eventual home of our large memory, is a very efficient computer in every respect. The ideas developed on TX-0 will be that much more powerful on TX-2.

The following is a list of subroutines in the system which may be useful in other programs written for TX-0:

I. Storage Print-out
   - A. Vertical column layout
     1. 2^6 words/column, 2 columns/page
     2. initial conditions
        - a. 173560 = +0 = instructions
             173560 = -0 = octal numbers
        - b. 173561 = first address of range to be printed
        - c. 173562 = last address of range
     3. Starting address of subroutine = 173323
   - B. Horizontal layout
     1. 2^3 words/line, 2^6 lines/page
     2. initial conditions
        - a. same as for Vertical Column Layout
     3. Starting address of subroutine = 172767

II. Computer Word Print-out
   - A. Instructions
     1. initial conditions
        - a. 177341 = word to be translated and printed
     2. Starting address of subroutine = 173140
   - B. Octal number (address)
     1. initial zero suppression

2. Initial conditions

a. 177337 = word to be translated and printed

3. Starting address of subroutine = 177370

C. Octal Number

1. All six digits are printed

2. Initial conditions

a. 177337 = word to be translated and printed out

3. Starting address of subroutine = 177710

III. English Word Printer

A. See appendix for list of English words available and their corresponding addresses.

B. The addresses of the words to be printed must be stored in the registers immediately following the transfer control instruction to this routine. The return address must be the address of the register following the register containing the address of the last English word to be printed.

C. Starting address of the subroutine = 173700

D. Example: print WHAT DO YOU WANT and a carriage return

WHAT = 174311
DO = 174234
YOU = 174237
WANT = 174243
CAR RETURN = 174231

Absolute Address Programming		Symbolic Programming	
100	cla	question,	cla
	add 200		add tn+answer
	trn 173700		trn English
	174311		w h a t
	174234		d o
	174237		y o u
	174243		w a n t
	174231		c a r r e t u r n
110	cla	answer,	cla
	etc		etc
	etc		etc
200	trn 110	English = 173700	

Note: The TX-0 Conversion Program written by Wes Clark knows the addresses of all the words in the vocabulary of the system. This is achieved by a Flexowriter dictionary tape. Each word on the tape has a space separating each letter so that should there be a redundancy between an English word to be processed by this subroutine, and one which the programmer is using as an address tag, the conversion program will not be confused; i.e., t e s t vs. test.

IV Computer Word Reader (HARK)

A. This subroutine will read one word terminated by a carriage return or tab.

B. The binary translation of the word will be in register 176737 when control is transferred back to the main program.

C. Starting address of the subroutine = 176333

V. Word Answer Reader (EXPEDITE)

A. This subroutine will read a YES or NO terminated by a carriage return or tab.

B. Register 173322 will contain a +0 for YES and a -0 for NO when control is returned to the main program.

C. Starting address of the subroutine = 173307

D. Other English answers must be programmed and their words added to the vocabulary of EXPEDITE.

VI. Storage Punch-out

A. Read-In Mode Format

1. Initial conditions

a. 172506 = first address of range to be punched out
b. 172510 = last address of range

2. Starting address of subroutine = 172041

B. Normal Input Routine Format

1. Initial conditions

a. 172504 = +0 = Input routine will be punched out first in Read-In Mode Format before range is punched out.
-0 = no Input Routine leader.

b. 172505 = first address of range to be punched out

c. 172507 = last address of range

2. Starting address of subroutine = 172100

C. Starting Instruction Punch-out

1. Initial conditions

a. 172511 = add (starting instruction address) if stop

= trn (starting instruction address) if no stop after read-in is desired.

2. Starting address of subroutine = 172203

D. Blank tape feed-out

1. This routine feeds out six inches of blank tape.

2. Starting address of subroutine = 172213

VII. Flow Chart Display and Sequence Record

A. Sequence reset subroutine

1. This routine is used when a new sequence is to be remembered. It is usually requested once at the beginning of a program.

2. Starting address of the subroutine = 171246

B. Flow Chart Display

1. This subroutine illuminates the boxes of a transparent copy of a flow diagram which is mounted on the face of the display scope. It also remembers the sequence in which the boxes were illuminated.

2. The three registers following each transfer control instruction to this subroutine must contain the following:

a. First reg = octal values of the x,y coordinates of the lower left corner of the box to be displayed. e.g., if x = 10, and y = 300, then 1st reg. = 010300.

b. Second reg = octal values of the width and height of the box to be displayed. e.g., if width = 70, and height = 50, then 2nd reg. = 070050.

c. Third reg = flexowriter code for letter or numeral used to identify the box being displayed. The code is written as a six digit octal number. e.g., if letter = r, 3rd reg.= 010100.

3. The return address must be the address of the fourth register following the transfer control instruction to this subroutine.

4. Starting address of the subroutine = 171000

5. Example: Consider the program flow chart of figure 4. Let us assume that we are going to write the request which will display the box that represents that section which does the indexing and compares XL and XK. The request is written in the program just before the instructions which actually do the indexing and comparing. The coordinates of the lower left corner of the box are -40, -310. The width of the box is 100 and the height is 70. The Flexowriter code for g is 110100. The request in instruction form would be:

```
100     cla
        add 200
        trn 171000
        737467
        100070
        110100
        cla
        etc.
        etc.
        etc.
        etc.
        etc.
        etc.
        etc.

200     trn 106
```

These instructions would contain the instructions for indexing the values and comparing XL and XK.

C. Sequence Print-out

1. This subroutine will print out the letters and numerals which represent the various sections of the program in the sequence that they were performed.
2. Using this routine, the sequence may be printed out at any time during the operation of the program. If it is not desirable to continue remembering the whole sequence then this routine should be followed by the Sequence Reset routine.
3. Starting address of the subroutine = 171257

PART IV Typing Directions

To use the Direct Input Utility System of TX-0, read in tape no. 18(latest mod). Be sure the TYPE IN switch is in the on position and start the program at register 175100. This will transfer control to EXPEDITE which is the title-read routine for the System. If you are not acquainted with the names of the other routines, type

expedite and a carriage return.

This will cause the computer to type out the list of the names.e.g.

expedite ↵

bebrief
word
address
find
hark
surprise
outlaw
punchy
prince
yes
no
expedite
me
r
path

If a name is typed incorrectly or is not in the above list, EXPEDITE will inform you of the fact. For example let us assume that you typed

be breif and a carriage return.

Spaces and upper and lower case are ignored but misspelling is not anticipated. Therefore EXPEDITE would have typed out the following:

error bebreif

HARK

To transfer control to the system's conversion program, type

hark and a carriage return.

Case 1. TO EXAMINE A REGISTER:

Type the address of the register, a vertical bar and an equal sign. HARK will type out the contents of the register as an octal instruction, a tab, and then wait for a modification. Since no modification is to be made, the typist will type a carriage return and procede to examine some other register. e.g. examine registers 174 and 300.

174| = add 177 TAB C.R.

typist HARK typist

300| = shr TAB C.R.

typist HARK typist

Case 2. EXAMINE AND MODIFY A REGISTER

Octal instructions, coded operate class commands, octal numbers, and any combination of these three separated by plus or minus signs may be used to express a modification. e.g. change register 174 to contain add 157.

174| = add 177 TAB add 157 C.R.

typist HARK typist

Case 3. MODIFY A REGISTER WITHOUT EXAMINING IT

174| add 157, C.R.

typist

Case 4. MODIFICATIONS OF A SERIES OF REGISTERS

```
174| add 157
sto 300
+234
```

In this case registers 174,175,and 176 would have been changed to add 157, sto 300, and +234. Each time a modification is typed and terminated the current address in HARK is indexed by one. Extra carriage returns and tabs are ignored.

Case 5. EXAMINATION AND MODIFICATION OF A SERIES OF REGISTERS

```
174| = add 177 ↵      add 157 ↵
typist  HARK          typist

175| = sto 301 ↵      sto 300 ↵
typist  HARK          typist

176| = sto 230 ↵      +234 ↵
typist  HARK          typist
```

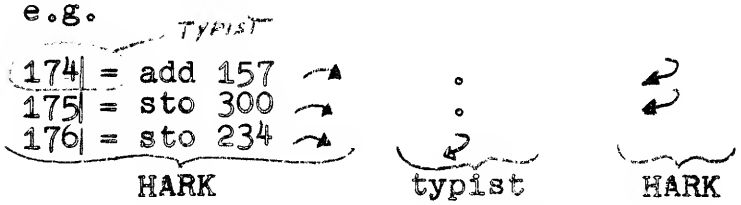
Using the carriage return to terminate a modification you can see that it is necessary for the typist to type the next address and an equal sign in order to obtain the examination. This is not necessary if the modification is terminated by a period. e.g.

```
174| = add 177 ↵      add 157.  ↵
typist  HARK          typist    HARK

175| = sto 301 ↵      sto 300.  ↵
HARK                  typist    HARK

176| = sto 230 ↵      +234.     ↵
HARK                  typist    HARK

177| = sto 400 ↵      ↵
HARK                  typist
```

The period will cause HARK to store the modification, produce the carriage return, type the next address, the vertical bar, the contents of the register, a tab, and then wait for another modification from the typist.

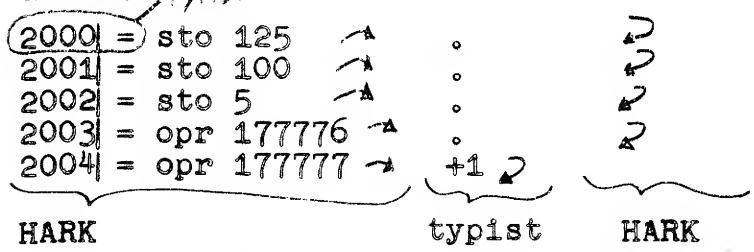
Case 6. EXAMINATION OF A SERIES OF REGISTERS

The period is also helpful when a series of registers is to be examined without modifications. When HARK receives a period from the typewriter it checks to see whether a modification has been typed. If a modification has not been typed HARK will merely produce a carriage return, type the next address, a vertical bar, the contents of the register, and a tab. e.g.



Case 7. A SEARCHING EXAMPLE

Using the period it is quite simple to search a given set of registers and then modify the desired one. For example if an entry in a table was to be changed it would not be necessary to know its exact location. One would merely type the first address of the table and search for the entry by typing periods. Of course if the table was extremely long the PRINCE would be used instead. Let us assume that we want to search the table beginning at register 2000 and that we want to change the first register containing a minus zero to a one.

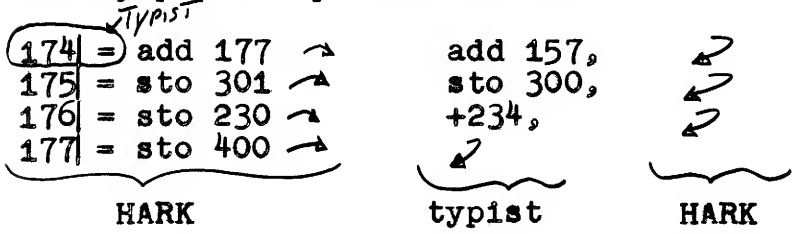


Case 8. EXAMINATION, MODIFICATION, AND RECORDING

The comma performs the same function as the period but also enables the typist to obtain a punched binary tape record of his modifications. When a modification is terminated by a comma HARK stores the mod in the indicated register, punches the binary form of the current address on tape, punches the binary form of the modification on tape, produces a carriage return, types the next address, a vertical bar, an equal sign, the contents of the register, and a tab.

Case 8. continued.

Let us assume that we had to make modifications to registers 174,175,and 176 and that we also wanted a binary paper tape record as well.



The paper tape would contain the following:

```
First three lines=    sto 174
next  three lines=    add 157
next  three lines=    sto 175
next  three lines=    sto 300
next  three lines=    sto 176
next  three lines=      +234
```

This kind of layout is called the Read-In format and is discribed in 6m-4789, A FUNCTIONAL DISCRIPTION OF THE TX-0 COMPUTER .

Case 9. EXAMINATION AND RECORDING OF A SERIES OF REGISTERS

This would be obtained by typing a comma in place of a period.

Case 10. RECORDING THE ADDRESS OF THE STARTING INSTRUCTION ON PAPER TAPE

After a series of modifications are made it is desirable to record the address of the starting instruction. This enables the Read-In mode of TX-0 to transfer control to the program after the modifications have been read into storage from tape. To do this the typist merely types the address of the starting instruction, a vertical bar, and a comma. e.g.

174| ,

This will cause the next three lines on paper tape to contain add 174(the add signals the Read-In mode to stop the computer before transfering control to register 174) and then HARK will feed out some blank tape.

Case 11. TRANSFERING CONTROL FROM HARK TO SOME OTHER REGISTER IN MEMORY

To transfer control to some other register in memory the typist has to type the address followed by two vertical bars. e.g.

174||

Case 12. CONCEALING INFORMATION FROM HARK

We have found that the typewritten copies produced while using Hark have been very useful records. There is a feature in HARK which allows the typist to make comments alongside any of the examinations or modifications without HARK acting on the information. Whenever the vertical bar is the first piece of information that HARK receives after a terminating character, then all the information that is typed will be ignored until a carriage return is typed. e.g.

174| = add 177 → add 157 →| add x is now add y ↵
TYPIST HARK TYPIST TYPIST

CASE 13. ERROR RECOGNITION BY HARK

Whenever an invalid word is typed Hark will inform the typist by typing the word "error" and the invalid word. Some examples of invalid words are:

1. Three letter combinations for which there are no operate class commands. i.e. clu,cpc,etc.
2. Any alphabetical letter used as a suffix to an octal number. i.e. 125j
3. The use of o and l for 0 and 1. i.e. 2oo and 17510.
4. Two letter designations for the four operation symbols. i.e. ad, st, tn, op, etc.

When HARK does find an invalid word it will not disturb the current address when it prints out the error.

Case 14. NULLIFY CONTROL

There are many times when a typist will make a mistake and discover it before a terminating character has been typed. To nullify the whole word all that is required is the typing of a double x. This will erase the whole word or address being typed but will not disturb the current address. This feature is also provided by EXPEDITE.

Case 15. HOW TO TRANSFER CONTROL BACK TO EXPEDITE FROM HARK

To transfer control back to EXPEDITE the typist merely has to hit the STOP CODE button on the typewriter. This will cause EXPEDITE to guard the typewriter for incoming requests for other routines.

Case 16. SOME EXAMPLES OF LEGAL WORDS

We mentioned that a word may be an octal number, an operate class command in coded form, an octal instruction, or any combination of these three so long as they were separated by plus or minus signs. Here are some examples:

1. add 100 +5 -add -110 will be converted to -5.
2. add+add will become trn 0
3. +2-1+100 will become +101
4. -add-100 will become (the complement of add 100 or) trn 177677.
5. cla +200 will become opr 140200

Note. The addition of two coded operate class commands will not become a new combination including the functions of the two commands that were added together...ie. cll+clr ≠ cla.
cll+clr = trn 140001.

6. 125+40| will cause the current address to be 165.
7. 125+40|| will cause HARK to transfer control to register 165.

SUMMARY OF TYPING DIRECTIONS FOR HARK

Current Address---------- address

Examine Current Address------ =

Octal Instructions---------- sto, add, opr, trn — alone or followed by an address

Octal Number---------- six or less octal digits alone, or preceded by a plus or minus sign.

Coded Operate Class Commands----------

cla	cll	clr	clc	cal
cyr	cyl	com	lac	alr
lpd	lro	lad	tac	tbr
shr	hlt	dis	ios	p7h
p6h	p6s	p7a	p6a	pnt
pna	pnc	rfa	rfl	rfr
r3c	r1c	r1l	r1r	

Terminating Characters:

1. Carriage return-------
2. Tab----------

will store modification in the register whose address is the current address and then index the current address by one. If there is no modification to be stored HARK will ignore the tab or carriage return and the current address will remain the same. Hark also ignores extra carriage returns or tabs.

3. Period---------- will store a modification in the same way as the tab and the carriage return do but in addition it will produce a carriage return, print the next current address, a vertical bar, an equal sign, the contents of the register whose address is now the current address, and a tab. If there is no modification then none will be stored. The rest of the sequence will remain the same.

SUMMARY OF TYPING DIRECTIONS FOR HARK continued.

Terminating Characters cont.

4. Comma---------- performs the same function as the period but in addition it will record the modification on paper tape in the format of the Read-In Mode. If no modification is made then it will record the original contents of the register whose address is the current address and then procede in same way as the period.

Additional Features

1. Recording the address of a starting instruction on paper tape---- address| ,

2. Transfering control to another section in memory---------- address| |

3. Concealing information from HARK----------| information |

4. Nullify control------- xx will nullify a whole word or address.

5. Transfering control back to EXPEDITE------ press the STOP CODE button.

Note: HARK will always type in the opposite color code of the typist.

IV IDEAS FOR NEW ROUTINES

As you can see, this utility system is very basic.

We have merely suggested a technique which will enable a programmer to see what his program is doing and examine and modify it rapidly while he is still at the console.

There are some new routines being written now which will be added to the system at a later date. When we have them written and operating we will write an addendum to this memo. The following is a list of some of the new routines and changes:

1. Several display subroutines
2. The flexo paper tape conversion program written by Wes Clark will be added to the system and will include the feature of informing the programmer of any illegal words on the flexo tape and asking him whether he wishes the conversion process to be continued with the illegal word (s) ignored or corrected. The correction would be added via the direct input flexo typewriter.
3. The direct input conversion routine, HARK, will have its vocabulary increased to understand relative and symbolic address tags.
4. In order to assure the programmer at the console that the utility system has not been damaged by his program, we plan on installing a check routine which will perform a sum check on all permanent instructions and constants in the system.
5. Any suggestions for routines or techniques which would increase the efficiency of debugging at the console will be appreciated.

John T. Gilmore, Jr.

JTG:bac

Attachments:

Appendix A	DL-1585
Appendix B	A-80287
B-80354	A-68405
D-80351	

APPENDIX A

CODED OPERATE CLASS COMMANDS

CLA = opr 140000	Clear the accumulator
CLL = opr 100000	Clear the left nine bits of the accumulator
CLR = opr 40000	Clear the right nine bits of the accumulator
CLC = opr 140040	Clear the accumulator and complement it
CAL = opr 140200	Clear the accumulator and live register
CYR = opr 600	Cycle the accumulator one position to the right
CYL = opr 31	Cycle the accumulator one position to the left
COM = opr 40	Complement the accumulator
LAC = opr 140022	Transfer the contents of the live register to the accumulator
ALR = opr 201	Transfer the contents of the accumulator to the live register
LPD = opr 22	Partial add the contents of the accumulator and the live register and leave result in AC
LRO = opr 200	Clear the live register
LAD = opr 32	Add the contents of the live register and the accumulator and leave result in AC
TAC = opr 140004	Transfer the contents of the toggle switch accumulator to the accumulator
TBR = opr 140023	Transfer the contents of the toggle switch buffer register to the accumulator
SHR = opr 400	Shift the accumulator to the right one position (multiply by 2^{-1})
HLT = opr 30000	Stop the computer
DIS = opr 22000	Display a point on the face of the oscilloscope according to the value in the accumulator
PEN = opr 100	Read light PEN into Ac_0 and Ac_1
IOS = opr 160000	In out stop and accumulator cleared
P7H = opr 27600	Punch seven holes on paper tape and cycle AC right one position (AC 2, 5, 8, 11, 14, 17 Tape 1 2 3 4 5 6)
P6H = opr 26600	Punch six holes on paper tape and cycle AC right
P6S = opr 166000	Clear AC and punch one line of blank tape
P7A = opr 27012	Punch seven holes and clear AC

CODED OPERATE CLASS COMMANDS (CONTINUED)

P6A = opr 26021	Punch six holes and clear AC
PNT = opr 24600	Print and cycle right (AC 2, 5, 8, 11, 14, 17 - Flexo 1, 2, 3, 4, 5, 6)
PNA = opr 24021	Print and leave AC cleared
PNC = opr 24061	Print and clear and complement AC
RFA = opr 141000	Clear AC and start petr running
RFL = opr 1031	Cycle AC left and start petr running
RFR = opr 1600	Cycle AC right and start petr running
R3C = opr 163000	Clear AC and read three lines from paper tape (Tape 1, 2, 3, 4, 5, 6 AC 0, 3, 6, 9, 12, 15)
R1C = opr 161000	Clear AC and read one line from paper tape
R1L = opr 161031	Clear AC, read one line and cycle left
R1R = opr 161600	Clear AC, read one line and cycle right

APPENDIX B

ENGLISH VOCABULARY of the TX-0 ENGLISH PRINTER SUBROUTINE

a	174170	outlaw	174444
address	174163	print	174212
after	174361	printed	174337
an	174413	program	174120
are	174100	punched	174113
area	174427	question	174455
be	174334	reading	174365
block	174104	reference	174416
car return (↵)	174231	routine	174271
column	174205	s	174160
comma	174410	search	174433
computer	174350	sir	174474
did	174466	starting	174142
do	174234	stop	174355
double	174173	tab (↷)	174110
find	174400	test	174131
finished	174135	the	174315
first	174321	there	174301
for	174440	this	174125
in	174372	title	174344
input	174265	to	174331
instruction	174147	trn	174451
is	174276	TX-0	174375
last	174325	vertical	174200
layout	174247	want	174243
more	174305	what	174311
no	174226	where	174462
normal	174260	word	174404
octal	174254	yes	174222
of	174155	you	174237
or	174424	your	174074
out	174216		

	(4096)	(8192)	(12, 288)	(16, 384)	(20, 480)	(24, 576)	(28, 672)	(32, 768)	(36, 864)	(40,960)	(45, 056)	(49, 152)	(53, 248)	(57, 344)	(61, 440)
000	10, 000	20, 000	30, 000	40, 000	50, 000	60, 000	70, 000	100, 000	110,000	120,000	130,000	140,000	150,000	160,000	170,000
1, 000	11, 000	21, 000	31, 000	41, 000	51, 000	61, 000	71, 000	101, 000	111, 000	121, 000	131, 000	141, 000	151, 000	161, 000	171, 000
2, 000	12, 000	22, 000	32, 000	42, 000	52, 000	62, 000	72, 000	102, 000	112, 000	122, 000	132, 000	142, 000	152, 000	162, 000	172, 000
3, 000	13, 000	23, 000	33, 000	43, 000	53, 000	63, 000	73, 000	103, 000	113, 000	123, 000	133, 000	143, 000	153, 000	163, 000	173, 000
4, 000	14, 000	24, 000	34, 000	44, 000	54, 000	64, 000	74, 000	104, 000	114, 000	124, 000	134, 000	144, 000	154, 000	164, 000	174, 000
5, 000	15, 000	25, 000	35, 000	45, 000	55, 000	65, 000	75, 000	105, 000	115, 000	125, 000	135, 000	145, 000	155, 000	165, 000	175, 000
6, 000	16, 000	26, 000	36, 000	46, 000	56, 000	66, 000	76, 000	106, 000	116, 000	126, 000	136, 000	146, 000	156, 000	166, 000	176, 000
7, 000	17, 000	27, 000	37, 000	47, 000	57, 000	67, 000	77, 000	107, 000	117, 000	127, 000	137, 000	147, 000	157, 000	167, 000	177, 000 177, 740 177, 777
(4, 095)	(8, 191)	(12, 287)	(16, 383)	(20, 479)	(24, 575)	(28, 671)	(32, 767)	(38, 863)	(40, 959)	(45, 055)	(49, 151)	(53, 247)	(57, 343)	(61, 437)	(65, 535)

TX-0 DIRECT INPUT UTILITY SYSTEM

TX-0 INPUT ROUTINE

TX-0 MEMORY REFERENCE CHART

FIG. 1

D-M035i

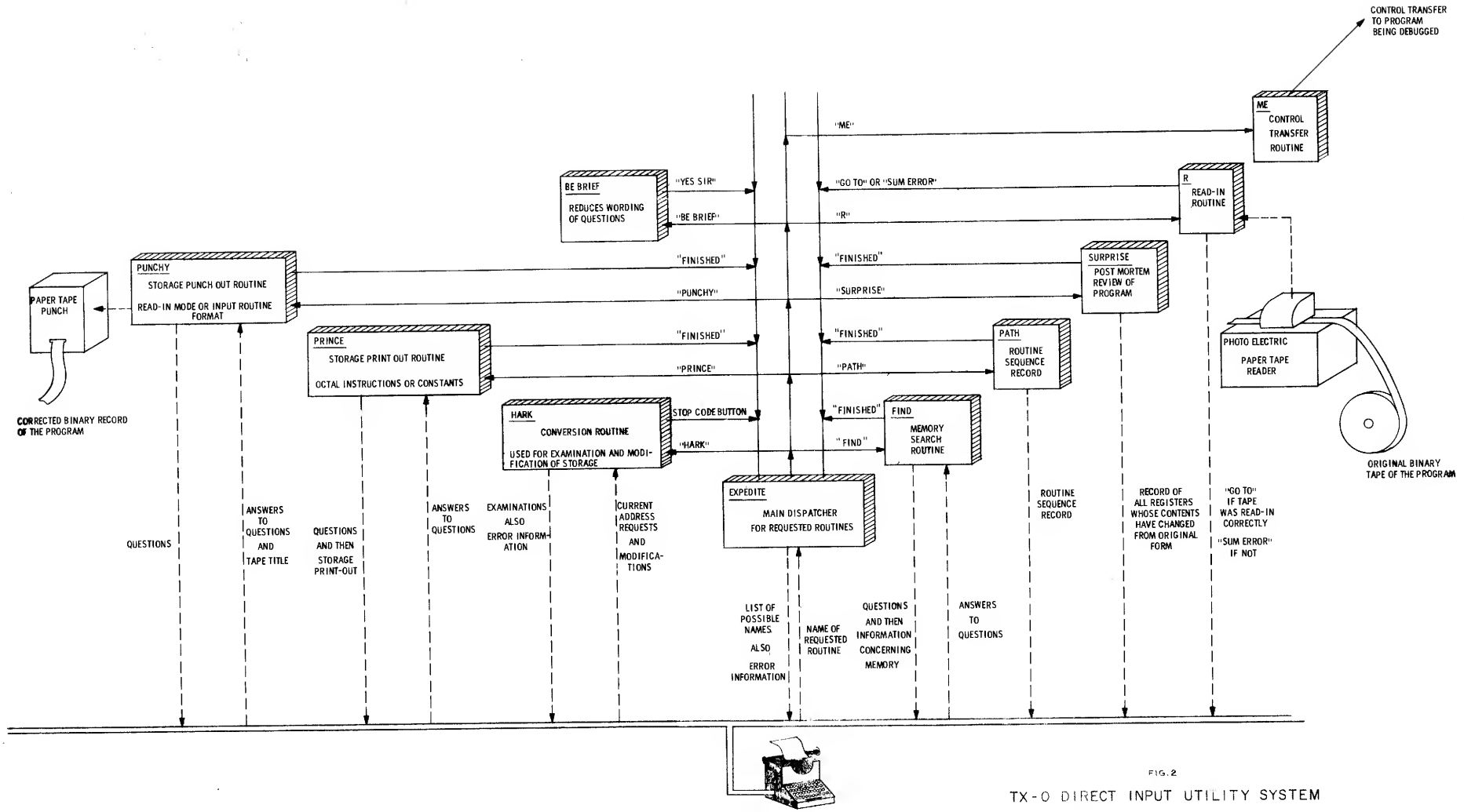
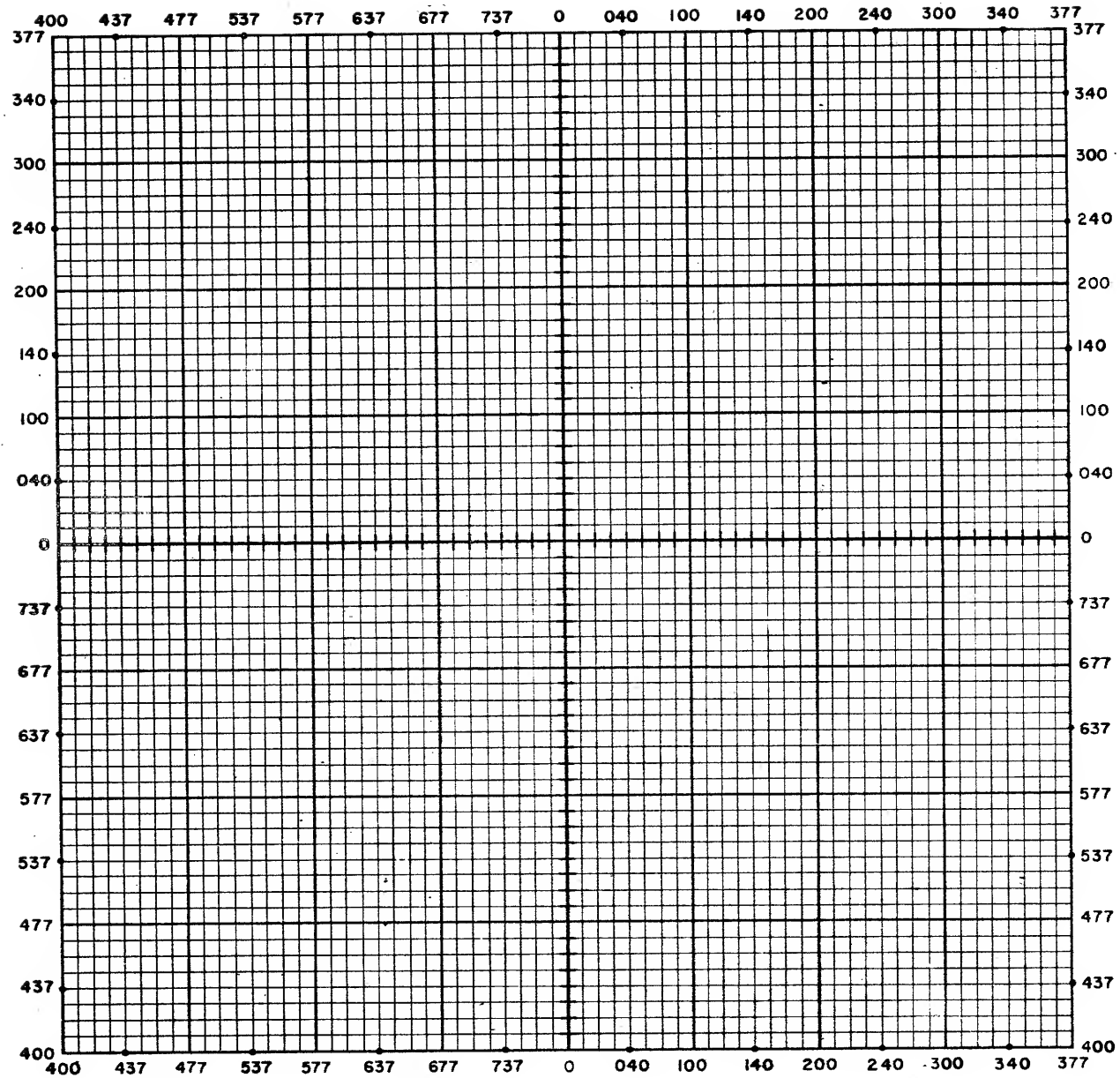


FIG. 2

TX-0 DIRECT INPUT UTILITY SYSTEM



TX-0 OCTAL GRAPH CHART

DL-1585
8-REDUCTION

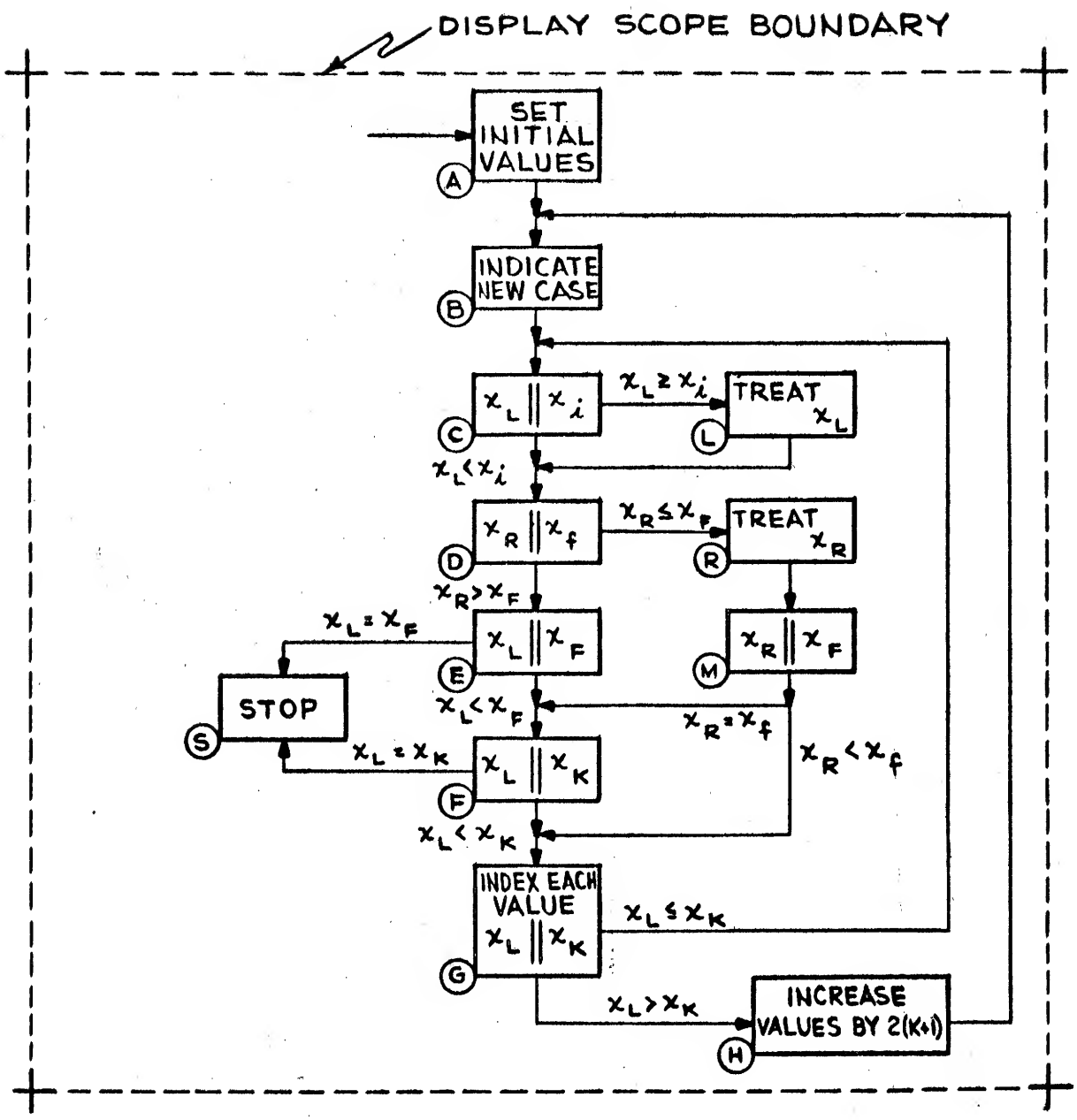
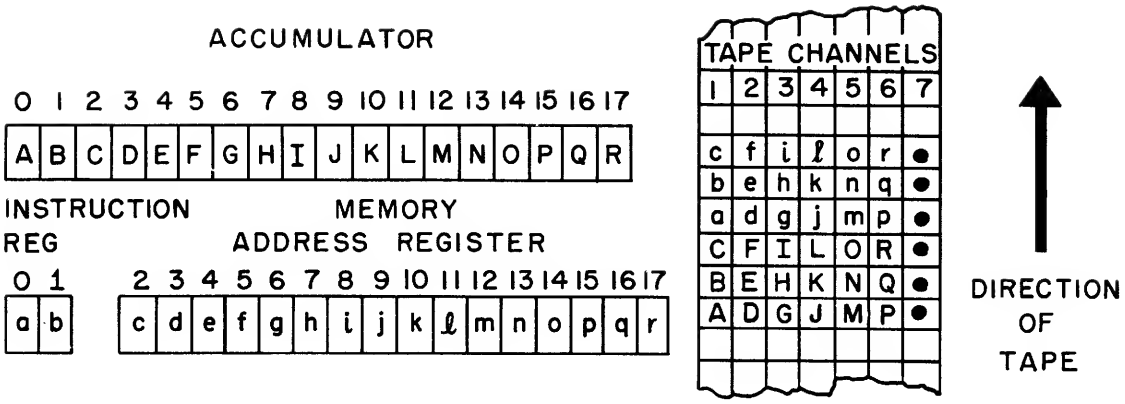


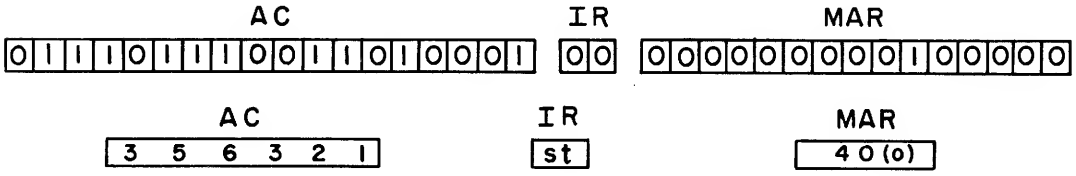
FIG. 4

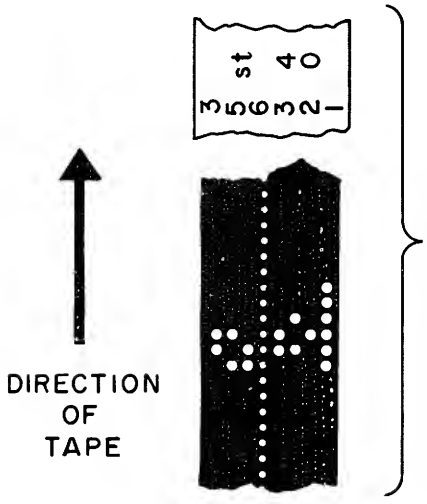
FLOW CHART EXAMPLE

A-80287



EXAMPLE: STORE THE OCTAL WORD 356321 IN REGISTER 40 OCTAL





IF THE TAPE IS HELD SO THAT IT IS MOVING FROM LEFT TO RIGHT WITH THE SEVENTH HOLE NEXT TO THE BODY, THE DATA WORD AND STORE INSTRUCTION CAN BE READ OCTALLY IN HORZONTAL FASHION.

FIG. 5

TAPE LAYOUT FOR READ-IN MODE OF TX-0

A-68405